EVOLUTION OF ADA TECHNOLOGY IN A PRODUCTION SOFTWARE ENVIRONMENT

Frank McGarry

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Md. 20771

Linda Esker

Computer Sciences Corporation
System Sciences Division
10110 Aerospace Rd.
Lanham-Seabrook, Md. 20706

Kelvin Quimby

Computer Sciences Corporation
System Sciences Division
10110 Aerospace Rd.
Lanham-Seabrook, Md. 20706

INTRODUCTION

The Ada programming language and the associated software engineering disciplines have been described as one of the most significant developments in software technology in many years. Although many claims have been made about its advantages and impacts, there have been very few empirical studies that clarify the impact of Ada.

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration (NASA) consisting of three principal members: NASA/Goddard Space Flight Center, the University of Maryland, and Computer Sciences Corporation. The SEL was founded in 1976 to carry out studies and measurements related to evolving software technologies [7]. The studies are aimed at understanding both the software development process and the impacts that evolving software practices may have on the software process and product. Since 1976, the SEL has conducted over 65 experiments by applying selected techniques to specific development efforts and measuring the resulting process and product.

In early 1985, the SEL initiated an effort to study the characteristics, applications, and impacts of Ada. Beginning with a relatively small practice problem (6000 source lines of Ada), the SEL has collected detailed development data from a total of eight Ada projects (some of which are still ongoing). The projects range in size from 6000 lines to approximately 160,000 lines of code.

PROJECT BACKGROUND

Development Environment

All Ada projects studied were developed in a DEC environment, using either a VAX 11/780 or a VAX-8600. Both machines are shared with other general users, and the support was average compared to other typical projects developed in FORTRAN. As will be pointed out later, varying degrees of use were made of the available tools and methodologies.

In studying the series of Ada projects, the goal/question/metric (GQM) paradigm [3] was followed. The goal of the study was to determine the impact of Ada on productivity, reliability, reuse, and general product characteristics. A second interest was to study the use of Ada features (such as generics and strong typing) over time.

Project History

Information on six Ada projects was analyzed for this study. The projects were developed over a span of 4-1/2 years starting in late 1984 and ending with two projects that will be completed in 1989. The study categorized the six projects into three groups distinguished solely by approximate start date: the first two are called the first Ada projects, the next two are the second Ada projects, and the most recent are the third Ada projects. The timeline for the six projects is shown in Figure 1.

The first experiences with Ada in the SEL occurred with two projects that were initiated in late 1984 and early 1985. A team of seven programmers was formed in late 1984 and began extensive training in December of 1985. The first target project was a simulator that was required to model an attitude control system of a particular NASA satellite, the Gamma Ray Observatory (GRO). Comparable simulators had been developed in the past by NASA, and this particular Ada project (GROOY) was developed in parallel to the identical project being developed in FORTRAN. The results of that particular comparison are documented by Agresti et al. [1] and McGarry and Nelson [14]. It was estimated that the development of GROOY would probably require from 10 to 12 staff-years of effort, considering previous experiences with similar FORTRAN projects.

The GROOY team had an average of nearly 5 years of experience with software development; however, none had any previous Ada experience. In fact, there had been no earlier Ada experience by anyone in this environment, so no lessons learned and no Ada experts were available to team members. The team was experienced with flight dynamics problems although it was, on the average, less experienced than the typical development teams in this environment.

To prepare for the design and development of GROOY, the team underwent 6 months of extensive

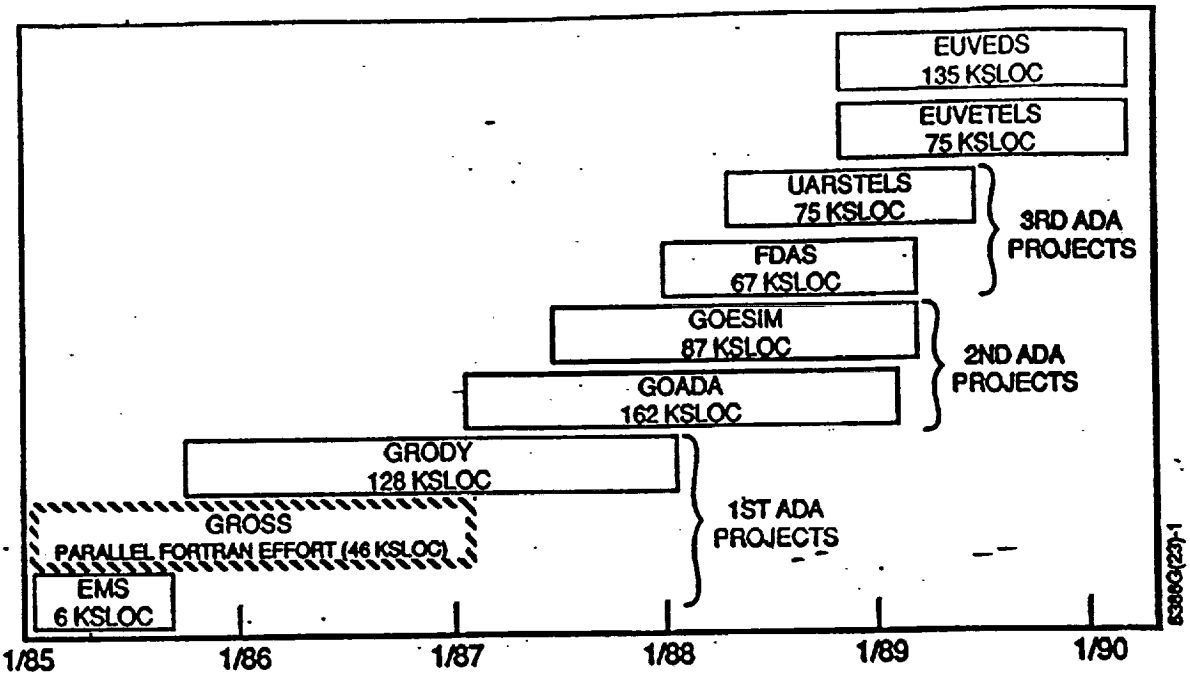

Figure 1. Ada Projects in the Flight Dynamics Division of NASA/Goddard

training, which covered Ada syntax as well as principles of software engineering and detailed design techniques [16]. The training included the following:

- Alsys video tapes with discussion
- Lectures on Ada from University of Maryland staff
- Lectures and workshops on PAMELA [8], object-oriented design, and other design techniques
- Workshops based on Booch's training materials [5]
- Lectures on software engineering principles, including abstraction and information hiding

During the 6 months of training, the electronic mail system (EMS) Ada project was developed. This was the first Ada effort completed by this organization. It was set up as a training problem, but detailed statistics were kept so that the development process and product could be analyzed. When completed, EMS consisted of approximately 6000 lines of Ada code.

These first Ada projects, GRODY and EMS, were developed by the same personnel in a similar software development environment. The EMS project was developed on a VAX 11/780 using the DEC Ada Compilation System (ACS); the GRODY project was developed on the VAX 8600, also using the DEC ACS.

The two projects classified as the second Ada projects both began in early 1987 and were of medium size with complexity typical of other efforts in this environment. Both projects were simulators required to support the GOES mission. GOADA was a dynamics simulator similar to GRODY; GOESIM was a telemetry simulator that would be used in testing the attitude ground support software.

The GOADA team consisted of approximately seven people, some of whom were not assigned full time to this project. Of these seven, three had previous Ada experience and two had experience in the application area. The GOESIM team consisted of four people, one of whom had previous Ada experience and one who had experience with this type of application. The training consisted of lectures and video tapes on Ada, with particular attention to the Ada style guide that had recently been developed; lectures and classes in Ada syntax and Ada concepts were also held. The training lasted approximately 4 weeks for each team.

Both of these second Ada projects used the DEC ACS to complete development. Because there had been previous Ada experience from the first Ada projects, experienced Ada programmers were available to these teams for consultation and guidance. They proved to be heavily used commodities. The second Ada projects spanned approximately 18 months each.

The final two projects analyzed in this study, the third Ada projects, were both started in early 1988. The UARSTELS project had requirements and characteristics similar to GOESIM, one of the second Ada projects. The other project, FDAS, was a much different type of system, a source code manager used for manipulating flight dynamics software components.

The UARSTELS team consisted of three people, one of whom had previous Ada experience; the FDAS team consisted of four people, none of whom had previous Ada experience. Training for both teams took the form of lectures and workshops based on

the Ada style guide; additional training from personnel with previous Ada experience was also provided. All team members attended classes on Ada syntax and Ada development.

The teams supporting the third Ada projects had more personnel available to them who had previous Ada experience, and they also had available several lessons-learned reports that had been developed by the earlier Ada projects. These two projects spanned approximately 15 to 18 months each.

Data Collection

Detailed data for this study were collected for the six projects. As for all software developed in the flight dynamics environment, the data are collected from the following sources:

- Data collection forms
- Tools and accounting records
- Interviews and subjective information

The most extensive and detailed data were provided on a series of data collection forms completed by both the developers and managers and including the following:

- Effort data (hours spent weekly by activity)
- Error data (for all changes and errors)
- Project estimation (managers' estimate of final size, cost, schedules)
- Product origination (characteristics of modules as they are designed)

These data were the major source of information used in the comparisons.

The tools used to record information include the automated accounting system (for records of computer time used, source code changes, and source code size history); configuration control tools (Configuration Management System (CMS), used to record changes to source code); and ASAP, the Ada Static Source Code Analyzer [9], which calculates detailed counts and characteristics of Ada source code. Information was also provided through interviews with team developers and managers (to record lessons learned and general impressions) and through subjective assessments made by senior software engineers (on topics such as methodologies applied). All this information is consistently collected, quality assured, and recorded on a data base where it is used as the basis for study or analysis.

EVOLVING CHARACTERISTICS OF THE SOFTWARE

Design Characteristics

The first Ada project did not automatically assume the reuse of the standard FORTRAN project methodologies and products. During the predesign phase, the project decided on the products, reviews, and methodology to be used. Project personnel decided to conform to the traditional design review process that had been used in the flight dynamics environment for many years [2, 15]. However, project members investigated several design methodologies and eventually applied a modified version of object-oriented design [1, 12, 19].

Use of an object-oriented design required a rethinking of the design, its documentation, and its presentation. This caused some inconsistencies with the traditional approach used in the SEL. Project members were required to develop new design products because the existing design documentation methods for structured and functionally oriented software design did not apply to the object-oriented design [17]. To develop a design notation used in the design documentation, they combined concepts from George Cherry's process abstraction method, PAMELA [8], and Grady Booch's object-oriented design [5]. Design diagrams were presented at the preliminary design review (PDR), and top-level package specifications were developed for the critical design review (CDR). These components were then expanded and compiled during Build 0 of the implementation phase.

The second and third Ada projects adopted and built on the same methodology and design notation. They also conducted design reviews, but the specific products generated at each stage were somewhat modified from the first Ada efforts. For these later projects, package specifications were developed for the PDR, and package bodies and subunit program design language (PDL) were developed for the CDR. In addition, the components were compiled before the CDR. These efforts pointed out the need to redefine the specific products produced at key milestones of the design process; however, the characteristics of the products have yet to be decided. Quimby and Esker [17] present a more detailed analysis of the evolving characteristics of both the design process and the products developed.

Software Size

Traditionally, software size has been described in terms of the lines of code developed for the system. Lines of code can, however, be expressed by many measurements [11], including the following:

- Total physical lines of code (carriage returns)
- Noncomment/nonblank physical lines of code
- Executable lines of code (ELOC) (not including declarations)
- Statements (semicolons in Ada, which include declarations)

Table 1 describes the size of the Ada projects in the flight dynamics environment using these four measurements. For comparison to the Ada projects, typical FORTRAN projects of similar applications are also summarized.

Unless only Ada statements are counted, these figures indicate that using Ada results in many more lines of code than using FORTRAN. The increase in lines of code is not necessarily a negative result; rather, it means that the size of the system implemented in Ada will be larger than an equivalent system in FORTRAN. It is also clear

Table 1. Software Characteristics of Projects

APPLICATION	1ST ADA PROJECTS	2ND ADA PROJECTS		3RD ADA PROJECTS		FORTRAN PROJECTS	
	DYN SIM	DYN SIM	TM SIM	CONFIG	TM SIM	DYN SIM	TM SIM
TOTAL LINES (CARRIAGE RETURNS)	128,000	162,200	87,500	67,700	75,000	45,500	28,000
NONCOMMENT/ NONBLANK	60,000	79,900	42,300	36,000	41,400	26,000	15,000
EXECUTABLE LINES (NO DECLARATIONS)	40,250	49,000	25,500	19,700	22,150	22,500	12,500
STATEMENTS (SEMICOLON – INCLUDES DECLARATIONS)	22,500	29,200	16,300	12,700	15,200	22,300	12,000

5388G(23)-5

Table 2. Effort Distribution (Percent of Total Effort) During Each Life-Cycle Phase of Ada and FORTRAN Projects

PHASE	1ST ADA PROJECTS	2ND ADA PROJECTS	3RD ADA PROJECTS	FORTRAN PROJECTS
REQUIREMENTS ANALYSIS	8	4.3	6.7	12.5
DESIGN	24	30.5	36.0	22.5
CODE	42	52.0	44.0	35.0
TEST	26	13.2	13.3	30

5388G(23)-6

that a precise definition is needed of what constitutes a line of code in Ada and what types of code are included in that measurement.

Many factors contribute to the increased size of the Ada projects. The style of Ada results in code growth because it encourages formatting, blank lines, and longer, readable names for data elements and subunits. The strong typing within Ada also produces more code than in FORTRAN because each data element must be explicitly declared. In addition, the local style guide places further requirements on the format for readability. Among other requirements, the style guide stipulates that each calling argument must be on a separate physical line. All these features have increased the code size, but the increased size also provides advancements in the areas of capability, readability, and understanding.

Effort Distribution by Phase Dates

Effort distributions can be described by the effort expended during the key life-cycle phases of a project and by the effort expended in software development activities. Using the first approach, effort distribution by phase dates, the typical FORTRAN life-cycle effort distribution [15] in the flight dynamics environment shows 12.5 percent of the total effort expended during the predesign or requirements analysis phase, 22.5 percent during the design phase, 35 percent during the code implementation phase, and 30 percent during the system test phase (Table 2).

From the review of literature on Ada [18], it was expected that the effort distributions would be significantly different for the Ada projects due to the modified design and implementation approaches. It had been anticipated that the Ada projects would require more effort during the detailed design phase and less effort during the code and test phases. However, in the flight dynamics environment, significant changes to the life cycle have not been observed. The Ada projects were planned by managers experienced with FORTRAN projects, and perhaps their plans were influenced by the FORTRAN life cycle.

Although the changes are not occurring as quickly as anticipated, the Ada life cycle is changing slightly with each project and may soon show a different life cycle than that expected for a FORTRAN project. The life cycles for the second and third Ada projects are shifting slightly to show more design time required and less system test time. The effort distributions of the Ada

Table 3. Effort Distribution (Percent of Total Effort) for Development Activities Across All Life-Cycle Phases of Ada and FORTRAN Projects

PHASE	1ST ADA PROJECTS	2ND ADA PROJECTS	3RD ADA PROJECTS	FORTRAN PROJECTS
REQUIREMENTS ANALYSIS	3.2	3.7	6.5	6.0
DESIGN	36.5	27.5	35.7	24.0
CODE	42.7	52.0	44.5	45.0
TEST	17.6	16.8	13.3	25.0

5386G(23)-7

projects are showing that the FORTRAN life cycle cannot be automatically assumed for Ada.

These observations probably indicate that the life-cycle definition is not easily changed merely because a different language or technique is applied. The evolution toward the expected characteristics of the new technology is a slow, gradual process.

Effort Distribution by Activity

The second approach to effort distributions is to analyze the effort by activity. In addition to collecting the effort expended between key phase dates (e.g., between the CDR and the start of system test), the SEL also collects detailed effort data independent of phase dates. Effort by phase is time driven and assumes, for example, that all design activities are complete and cease at the end of the design phase. In reality, many activities take place during each life-cycle phase and, therefore, the effort distribution by activity can be quite different from the distribution by phases.

This, indeed, was the case for the first Ada projects. Although only 24 percent of the total effort was allocated to the design phase, 36.5 percent of the total project effort was spent on design activities (Tables 2 and 3). The extra effort needed for Ada design activities is more apparent in the distribution of effort by activity than in the distribution by phase dates. The effort by development activity again reinforces the trend seen above. On the Ada projects, more effort was required for design and less effort for software testing than on the FORTRAN projects.

Use of Ada Features

In an effort to achieve some measurement of the use of the features available in the Ada language, the SEL identified six Ada features to monitor: generic packages, type declarations, packages, tasks, compilable PDL, and exception handling. The SEL then examined the code to see how little or how much these features were used. The purposes of this analysis were, first, to determine to what degree features of Ada were used by the Ada project and, second, to determine whether the use of Ada features "matured" as an environment gained experience with the language. Data on the use of these Ada features were obtained using the Ada Static Source Code Analyzer Program [9]. Analysis of the use of compilable PDL and exception handling did not show any trends. Perhaps it is too early to see results in these areas; however, trends were observed in the use of the other features.

The average size of packages (in source lines of code (SLOC)) for the first Ada projects is much larger than the average size of packages for the second and third Ada projects (Figure 2). This increase is due to a difference in the structuring method between the first Ada projects and all subsequent Ada projects [17]. The first Ada projects were designed with one package at the root of each subsystem, which led to a heavily nested structure. In addition, nesting of package specifications within package bodies was used to control package visibility. Current Ada projects are using the view of subsystems described by Grady Booch [6, Ch. 17] as an abstract design entity whose interface is defined by a number of separately compilable packages, and the only nested Ada packages are generic package instantiations.

The generic package is a major tool in the Ada language contributing to software reusability. Reports have shown the benefits of Ada reusable software [18] and, in the flight dynamics environment, use of generic packages has been increasing from the first to the current Ada projects. More than one-third of the packages on current projects are generic packages. Although more analysis is needed, this higher use of generics possibly reflects both a stronger emphasis on the development of verbatim reusable components and an increased understanding of how to use generic Ada packages effectively in the flight dynamics environment.

The use of strong typing in these software systems was measured by the number of type declarations per thousand lines of code. Although the measure itself is not intuitively meaningful, it provides a method of observing trends in the use of Ada type declarations. In the flight dynamics environment, the amount of typing is increasing over time. This may indicate that the developers are becoming more comfortable with the strong typing features of Ada and are using its capabilities to a fuller extent.

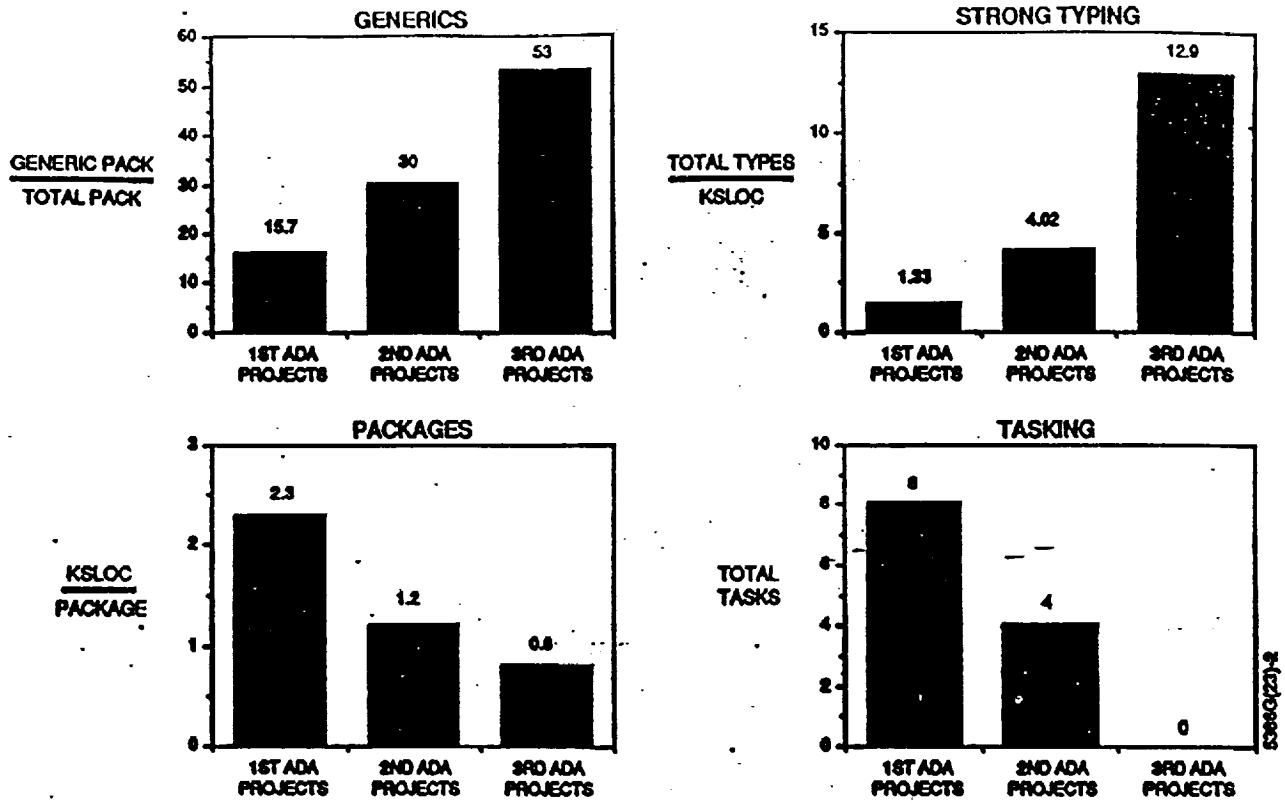

Figure 2. Use of Ada Features

The tasking feature of Ada is used in the flight dynamics environment for the dynamics simulators. Eight tasks were used for GRODY, the first dynamics simulator in Ada. For subsequent dynamics simulators of approximately the same size and functionality as GRODY, design personnel determined that four tasks were sufficient to implement the interactive capability of the system. It is expected that future dynamics simulators will continue to use tasking, however developers are now using tasking more judiciously. The third Ada projects are telemetry simulators, which are sequential systems that do not benefit from the feature of the language, and thus do not use Ada tasks.

COST/RELIABILITY/REUSE

Productivity

Discussions on Ada productivity require careful interpretation because so many definitions exist for software size measures in Ada. Depending on the measurement used, software developers using Ada can be shown to be either as productive as or not as productive as software developers using FORTRAN. Using the total lines of delivered code as a measure, the Ada projects studied show an improving productivity over time, and they show a productivity greater than FORTRAN (Figure 3). However, considering only code statements (semicolons for Ada or excluding all comments and continued lines of code in FORTRAN), the results are different. An increasing productivity trend remains in the Ada projects over time, but the Ada projects have not yet achieved the productivity level of FORTRAN projects.

In the flight dynamics environment, many software components are reused on FORTRAN projects. Because no Ada components existed previously, the first Ada projects were, in fact, developing a greater percentage of their delivered code than the typical FORTRAN project. A past study by the SEL and experience with FORTRAN projects indicated that reused code costs approximately 20 percent of the cost of new code [2]. Using this estimate, reusability can be factored into software size by estimating the amount of developed code. Developed code is calculated as the amount of new code plus 20 percent of the reused code. With software reusability factored in, the productivity for developed statements on Ada projects is approximately the same as that for FORTRAN projects (Figure 4).

Many objections are often made when computing productivity in terms of lines of code: it is affected by style, there are many ways to code the same function, etc. The most intuitive measure to use in computing productivity is cost per "function." Some attempts have been made within the SEL to compare the functionality of projects being compared (e.g., GRODY versus GROSS), and data seem to indicate that comparing "statements" is the closest measure to comparing functionality.

The trends in Ada productivity are very positive in that the overall cost of producing an Ada

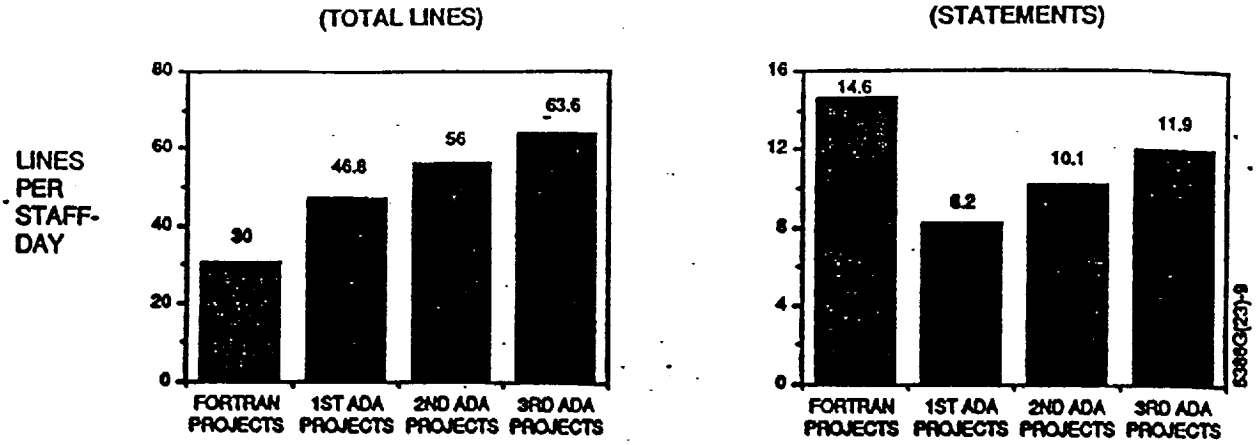

Figure 3. Ada Cost/Productivity of Delivered Code

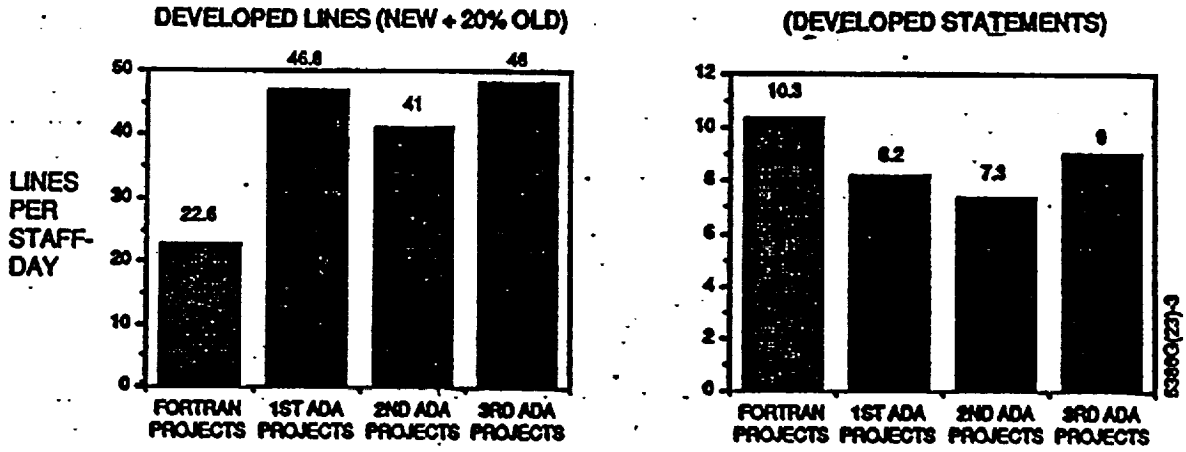

Figure 4. Ada Cost/Productivity of Developed Code

system has quickly become equivalent to that of producing a FORTRAN system. The flight dynamics FORTRAN environment is stable, mature, and built on a long and extended legacy of experience. Although the first Ada project required 30 percent more staff-hours to complete than a similar FORTRAN project, this overhead included effort to develop new practices and processes and to learn a new environment. With experience, the environment is becoming more stable and productivity is increasing.

Reliability

As with productivity, the many ways of measuring software size affect the results of reliability studies. For example, will the error rate normalized by the total system size or by the number of language statements give the most accurate reading of the reliability of Ada software compared to FORTRAN? In the flight dynamics environment, changes to the software made after unit testing when the software is placed under configuration control are formally reported on change report forms. The developer must supply the reason for the change (e.g., error, requirement change) and, if the change is due to an error, the source and type of error.

In a very mature FORTRAN development environment, the GROSS project reported 3.4 errors per thousand lines of source code (KSLOC) in the system. As Table 4 shows, all the Ada projects achieved an error rate lower than the rate on the FORTRAN project. In addition, the error rate on the Ada projects shows a decreasing trend over time.

When the error rate is normalized by the number of language statements, the first and second Ada projects show a slightly higher error rate than the FORTRAN project. However, the error rate again shows a decreasing trend over time. On the third Ada projects, the errors have decreased to a rate as good as, if not better than, the error rate on the FORTRAN project. It is still too early to observe a definite difference from the FORTRAN rates; however, the reliability of the Ada projects appears at least as good as that of FORTRAN projects and is improving with each Ada project.

Classes of Errors

Errors reported are classified according to source and type of error. Sources of errors can be requirements, functional specifications, design, code, or previous changes. Types of errors

Table 4. Ada and Error/Change Rate

	1ST ADA PROJECTS	2ND ADA PROJECTS**		3RD ADA PROJECTS**		FORTRAN PROJECTS
ERRORS/KSLOC*	1.8	1.7	1.4	1.0	1.0	3.4
ERRORS/K STMTS	10.2	9.4	7.5	5.3	5.6	6.9

5386G(23)-8

* SLOC = TOTAL LINES (INCLUDES COMMENTS/REUSED)
** FIGURES BASED ON ESTIMATES

are initialization, logic/control structure, internal interface, external interface, data value or structure, and computational.

On a typical FORTRAN project in the flight dynamics environment, design errors amount to only 3 percent of the total errors on the project (Figure 5). For the first and second Ada projects, 25 to 35 percent of all errors were classified as design errors, a substantial increase. For the third Ada project, however, design errors dropped significantly and are estimated to be approximately 7 percent. This rate is close to that experienced on FORTRAN projects and clearly shows a maturation process with growing expertise in Ada.

The literature on Ada reports that the use of Ada should help reduce the number of interface errors in the software [4]. Although the compiler will catch most calling parameter consistency errors, interface errors can also include errors that will not be detected until run time. Typically, these are errors in string parameters or subtypes with different constraints and errors in calling parameters due to the need for additional or different types of parameters. Using guidelines and examples in the data collection document [13], the errors are classified by the developer reporting and correcting the error.

In the flight dynamics FORTRAN environment, about one-third of all errors on a project are interface errors. On the first and second Ada projects, the percentage of interface errors was not greatly reduced (Figure 5), with approximately one-fourth of the errors being interface errors. With current projects, however, the SEL is now observing a significant change: interface errors are decreasing.

In the SEL, "errors due to a previous change" categorizes errors caused by a previous modification to the software. The first Ada projects showed a large jump in the percentage of these errors compared to projects using FORTRAN (Figure 5). However, all subsequent Ada projects show a rate for these errors that is very similar to the FORTRAN rate. This initial jump in the error rate can probably be attributed to inexperience with Ada, inexperience with Ada design methodologies, and a nested software architecture that made the software much more complex. Again, the error profile is evolving with the maturity of the Ada environment.

Software Reuse

Throughout the years of developing similar systems in FORTRAN in the flight dynamics environment, the average level of software reuse has been between 15 and 20 percent [10, 20]. FORTRAN projects that attained a software reuse rate of 35 percent or higher are rare. After the first Ada projects and with only 5 to 6 years of maturing in the environment, Ada projects have now achieved a software reuse rate of over 25 percent, already greater than the typical FORTRAN project. The UARSTELS project is expected to consist of more than 35 percent reused code. This trend of increasing software reuse is very promising.

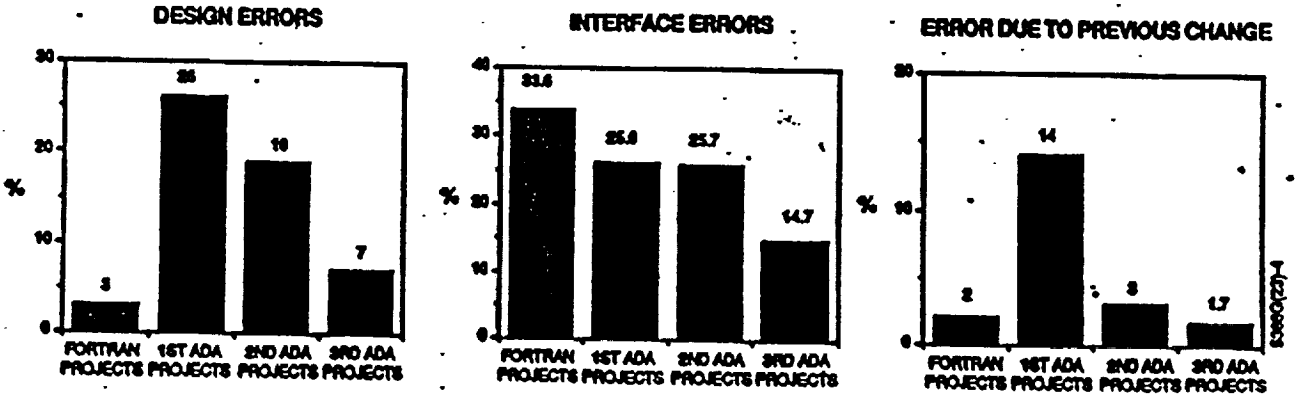

Figure 5. Error Characteristics

CONCLUSIONS/OBSERVATIONS/COMPARISONS

Many aspects of software development with Ada have evolved as our Ada development environment has matured and our personnel have become more experienced in the use of Ada. The SEL has seen differences in the areas of cost, reliability, reuse, size, and use of Ada features.

A first-time Ada project can be expected to cost about 30 percent more than an equivalent FORTRAN project. However, the SEL has observed significant improvements over time as a development environment progresses to second and third uses of Ada.

The reliability of Ada projects is initially similar to that expected in a mature FORTRAN environment. With time, however, improvements can be expected as experience with the language increases.

Reuse is one of the most promising aspects of Ada. The proportion of reusable Ada software on our Ada projects exceeds the proportion of reusable FORTRAN software on our FORTRAN projects. This result was noted fairly early in our Ada projects, and our experience shows an increasing trend over time.

The size of an Ada system will be larger than a similar system in FORTRAN when considering SLOC. Size measurements can be misleading because different measurements reveal different results. Ratios of Ada to FORTRAN range from 3 to 1 for total physical lines to 1 to 1 for statements.

The use of Ada features definitely evolves with experience. As more experience is gained, some Ada features may be found to be inappropriate for specific applications. However, the lessons learned on an earlier project play an invaluable part in the success of later projects.

REFERENCES

1. Agresti, W., et al. Designing with Ada for satellite simulation: A case study. Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986.

2. Agresti, W., McGarry, F., et al. Manager's Handbook for Software Development. Software Engineering Laboratory, SEL-84-001, April 1984.

3. Basili, V. Quantitative Evaluation of Software Methodology. University of Maryland, Technical Report TR-1519, 1985.

4. Basili, V., et al. Use of Ada for FAA's Advanced Automation System (AAS). The MITRE Corporation, April 1987.

5. Booch, G. Software Engineering With Ada. Menlo Park, CA: Benjamin/Cummings Publishing Company, 1983.

6. Booch, G. Software Components With Ada — Structures, Tools, and Subsystems. Menlo Park, CA: Benjamin/Cummings Publishing Company, 1987.

7. Card, D., McGarry, F., Page, G., et al. The Software Engineering Laboratory. Software Engineering Laboratory, SEL-81-104, February 1982.

8. Cherry, G. Advanced Software Engineering With Ada—Process Abstraction Method for Embedded Large Applications. Language Automation Associates, Reston, VA, 1985.

9. Doubleday, D. ASAP: An Ada Static Source Code Analyzer Program. University of Maryland, Department of Computer Science, Technical Report 1895, August 1987.

10. Esker, L. Software Reuse Profile Study of Recent FORTRAN Projects in the Flight Dynamics Area. Computer Sciences Corporation, IM-88/083(59 253), January 1989.

11. Firesmith, D. Mixing apples and oranges: Or what is an Ada line of code anyway? Ada Letters, September/October 1988.

12. Godfrey, S., and Brophy, C. Assessing the Ada Design Process and Its Implications: A Case Study. Software Engineering Laboratory, SEL-87-004, July 1987.

13. Heller, G. Data Collection Procedures for the Rehosted SEL Database. Software Engineering Laboratory, SEL-87-008, October 1987.

14. McGarry, F., and Nelson, R. An Experiment With Ada—The GRO Dynamics Simulator. NASA/GSFC, April 1985.

15. McGarry, F., Page, G., et al. Recommended Approach to Software Development. Software Engineering Laboratory, SEL-81-205, April 1983.

16. Murphy, R., and Stark, M. Ada Training Evaluation and Recommendations. Software Engineering Laboratory, SEL-85-002, October 1985.

17. Quimby, K., and Esker, L. Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis. Software Engineering Laboratory, SEL-88-003, 1988.

18. Reifer, D. Ada's impact: A quantitative assessment. Proceedings of the 1987 ACM SIGAda International Conference. December 1987.

19. Seidewitz, E., and Stark, M. General Object-Oriented Software Development. Software Engineering Laboratory, SEL-86-002, August 1986.

20. Solomon, O., and Agresti, W. Profile of Software Reuse in the Flight Dynamics Environment (Preliminary). Computer Sciences Corporation, CSC/TM-87/6062, 1987.